ЛАБОРАТОРНА РОБОТА № 4

ДОСЛІДЖЕННЯ МЕТОДІВ АНСАМБЛЕВОГО НАВЧАННЯ ТА СТВОРЕННЯ РЕКОМЕНДАЦІЙНИХ СИСТЕМ

Мета роботи: використовуючи спеціалізовані бібліотеки та мову програмування Python дослідити методи ансамблів у машинному навчанні та створити рекомендаційні системи.

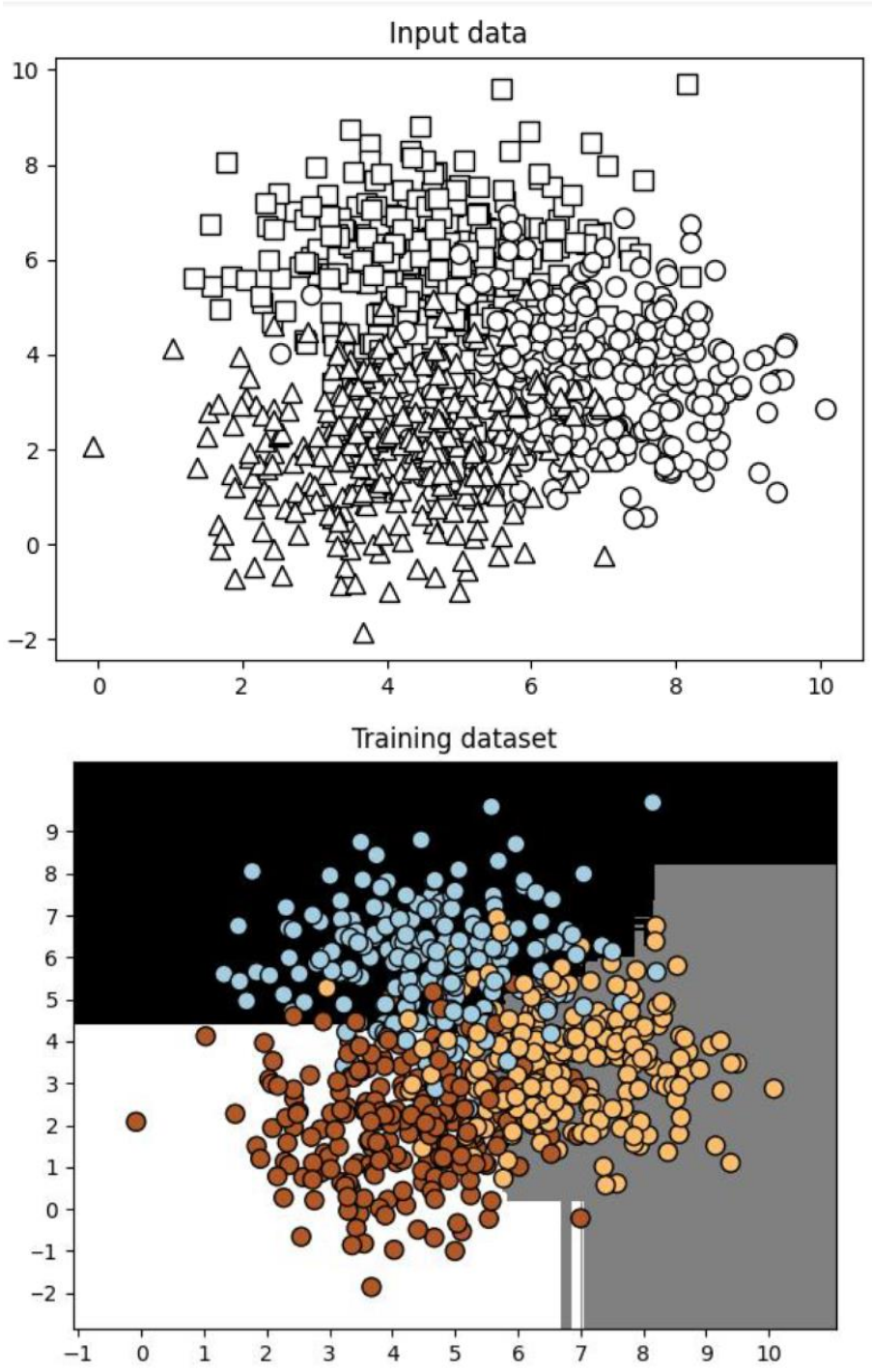
Завдання 4.1 Створення класифікаторів на основі випадкових та гранично випадкових лісів.

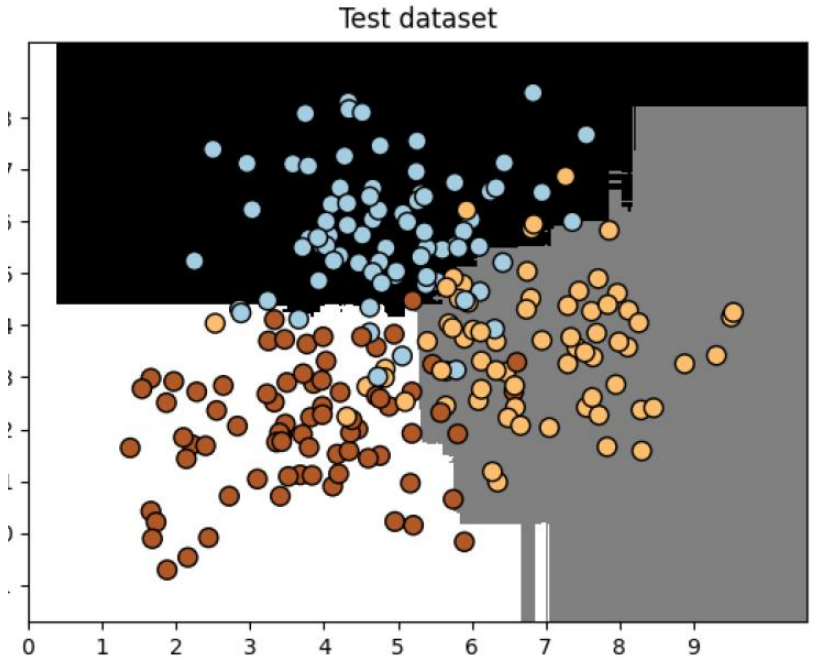
```python
import argparse
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier
from sklearn.metrics import classification_report
from utilities import visualize_classifier
# Парсер аргументів
def build_arg_parser():
    parser = argparse.ArgumentParser(description='Classify data using Ensemble Learning techniques')
    parser.add_argument('--classifier-type', dest='classifier_type',
                        required=True, choices=['rf', 'erf'],
                        help="Type of classifier to use; can be either 'rf' or 'erf'")
    return parser
if __name__ == '__main__':
    #Вилучення вхідних аргументів
    args = build_arg_parser().parse_args()
    classifier_type = args.classifier_type
    #Завантаження вхідних даних
    input_file = 'data_random_forests.txt'
    data = np.loadtxt(input_file, delimiter=',')
    X, y = data[:, :-1], data[:, -1]
    #Розбиття вхідних даних на три класи
    class_0 = np.array(X[y == 0])
    class_1 = np.array(X[y == 1])
    class_2 = np.array(X[y == 2])
    #Візуалізація вхідних даних
    plt.figure()
    plt.scatter(class_0[:, 0], class_0[:, 1], s=75, facecolors='white',
```

```python
                edgecolors='black', linewidth=1, marker='s')
    plt.scatter(class_1[:, 0], class_1[:, 1], s=75, facecolors='white',
                edgecolors='black', linewidth=1, marker='o')
    plt.scatter(class_2[:, 0], class_2[:, 1], s=75, facecolors='white',
                edgecolors='black', linewidth=1, marker='^')
    plt.title('Input data')
    #Розбивка даних на навчальний та тестовий набори
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.25, random_state=5)
    # Класифікатор на основі ансамблевого навчання
    params = {'n_estimators': 100, 'max_depth': 4, 'random_state': 0}
    if classifier_type == 'rf':
        classifier = RandomForestClassifier(**params)
    else:
        classifier = ExtraTreesClassifier(**params)

    classifier.fit(X_train, y_train)
    visualize_classifier(classifier, X_train, y_train, 'Training dataset')
    y_test_pred = classifier.predict(X_test)
    visualize_classifier(classifier, X_test, y_test, 'Test dataset')
    #Перевірка роботи класифікатора
    class_names = ['Class-0', 'Class-1', 'Class-2']
    print("\n" + "#" * 40)
    print("\nClassifier performance on training dataset\n")
    print(classification_report(y_train, classifier.predict(X_train), target_names=class_names))
    print("#" * 40 + "\n")
    print("#" * 40)
    print("\nClassifier performance on test dataset\n")
    print(classification_report(y_test, y_test_pred, target_names=class_names))
    print("#" * 40 + "\n")
    #Обчислення параметрів якості
```

```python
#Обчислення параметрів довірливості
test_datapoints = np.array([[5, 5], [3, 6], [6, 4], [7, 2], [4, 4], [5, 2]])
print("\nConfidence measure:")
for datapoint in test_datapoints:
    probabilities = classifier.predict_proba([datapoint])[0]
    predicted_class = 'Class-' + str(np.argmax(probabilities))
    print('\nDatapoint:', datapoint)
    print('Predicted class:', predicted_class)
    #Візуалізація точок даних
visualize_classifier(classifier, test_datapoints, [0] * len(test_datapoints), 'Test datapoints')
plt.show()
```

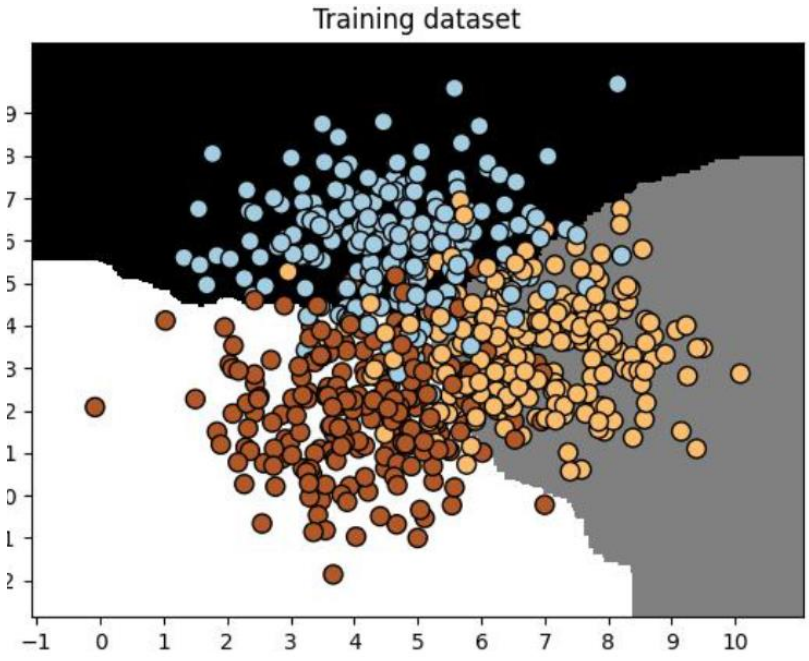
Рис 1. Код програми файлу LR_4_task_1

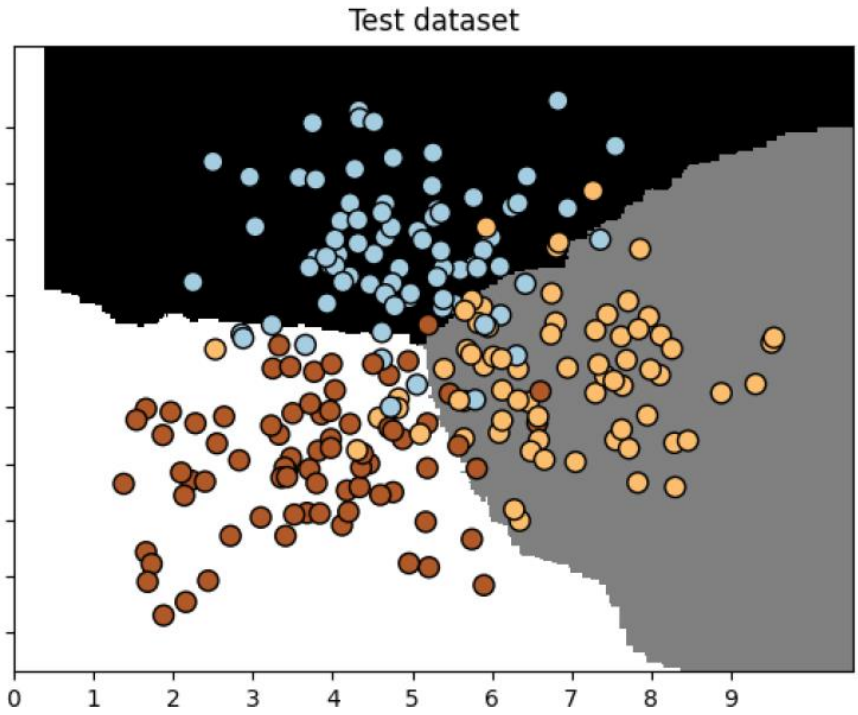




Class-0	0.92	0.85	0.88	79
Class-1	0.86	0.84	0.85	70
Class-2	0.84	0.92	0.88	76
accuracy			0.87	225
macro avg	0.87	0.87	0.87	225
weighted avg	0.87	0.87	0.87	225

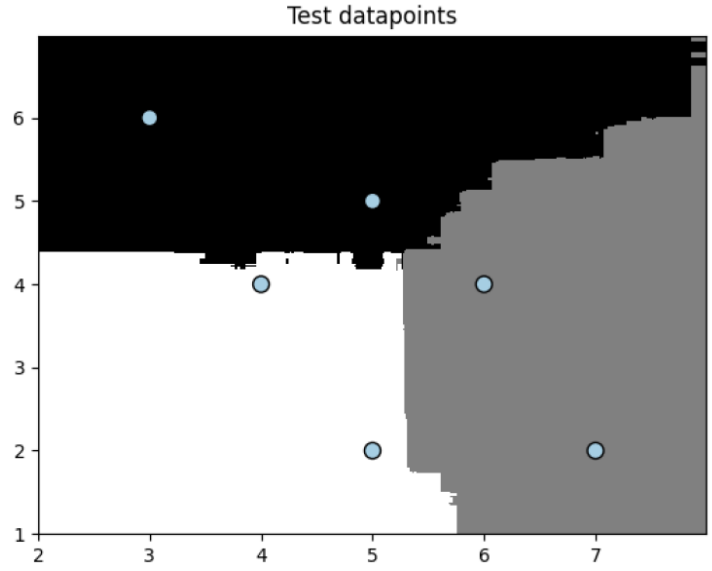
##################################



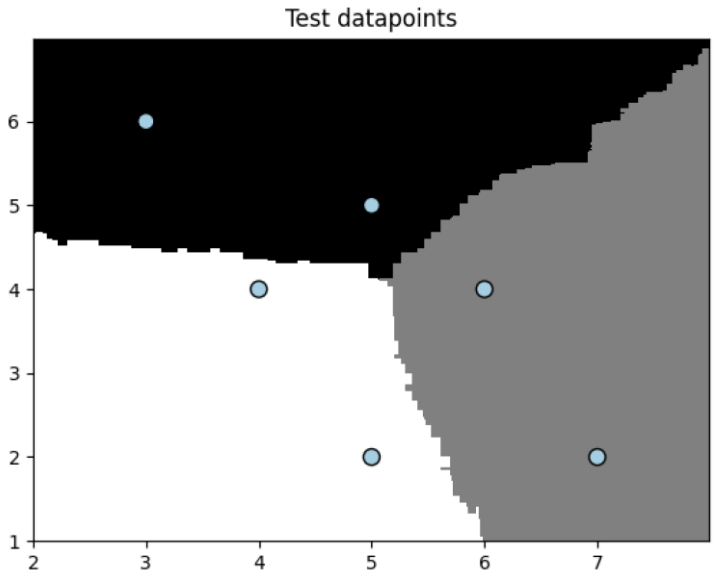


Classifier performance on test dataset

	precision	recall	f1-score	support
Class-0	0.92	0.85	0.88	79
Class-1	0.84	0.84	0.84	70
Class-2	0.85	0.92	0.89	76
accuracy			0.87	225
macro avg	0.87	0.87	0.87	225
weighted avg	0.87	0.87	0.87	225



```
Confidence measure:

Datapoint: [5 5]
Predicted class: Class-0

Datapoint: [3 6]
Predicted class: Class-0

Datapoint: [6 4]
Predicted class: Class-1

Datapoint: [7 2]
Predicted class: Class-1

Datapoint: [4 4]
Predicted class: Class-2

Datapoint: [5 2]
Predicted class: Class-2
```



```
Confidence measure:

Datapoint: [5 5]
Predicted class: Class-0

Datapoint: [3 6]
Predicted class: Class-0

Datapoint: [6 4]
Predicted class: Class-1

Datapoint: [7 2]
Predicted class: Class-1

Datapoint: [4 4]
Predicted class: Class-2

Datapoint: [5 2]
Predicted class: Class-2
```

Рис 2. Результат виконання коду файлу LR_4_task_1

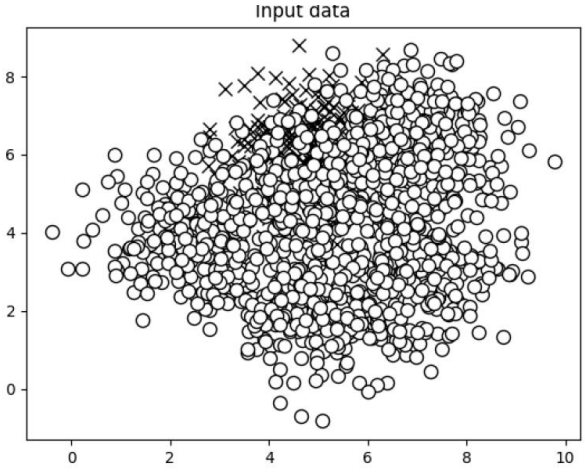
Висновок: При юзs **-erf** отримав більш валідні піки. Це через те, що в процесі навчання гранично випадкові ліси мають більше можливостей для вибору оптимальних дерев рішень, одже, як правило, вони забезпечують отримання кращих границь. Але кінцеві результати виявилися майже однаковими при використанні обох прапорців.

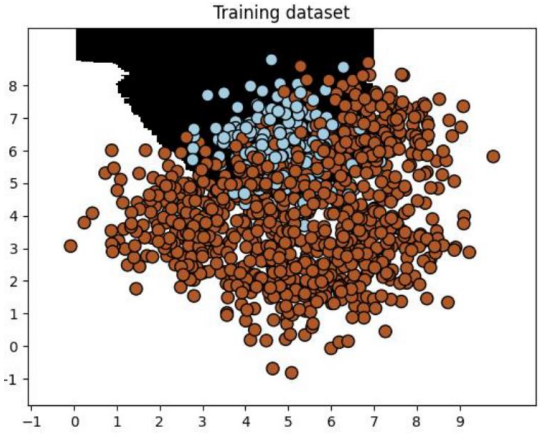
Завдання 2.2. Обробка дисбалансу класів.

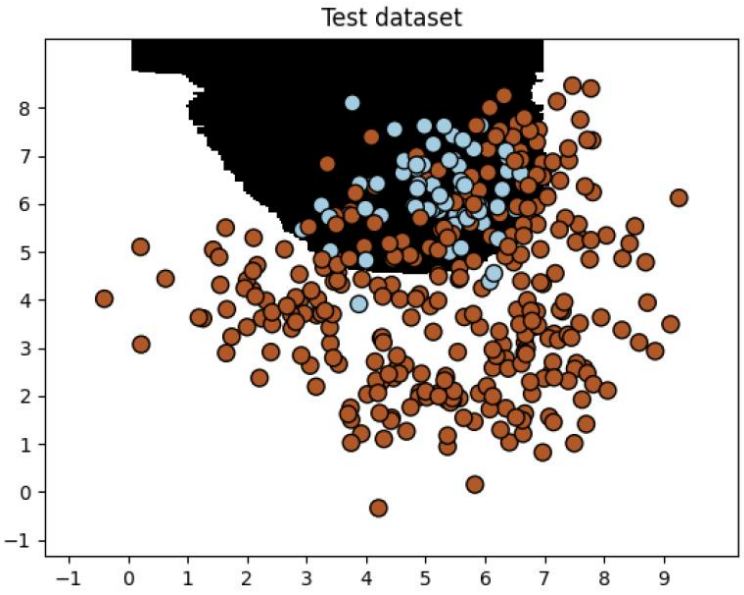
```python
import sys
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from utilities import visualize_classifier
# Завантаження вхідних даних
input_file = 'data_imbalance.txt'
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1]
# Поділ вхідних даних на два класи на підставі міток
class_0 = np.array(X[y == 0])
class_1 = np.array(X[y == 1])
# Візуалізація вхідних даних
plt.figure()
plt.scatter(class_0[:, 0], class_0[:, 1], s=75, facecolors='black',
            edgecolors='black', linewidth=1, marker='x')
plt.scatter(class_1[:, 0], class_1[:, 1], s=75, facecolors='white',
            edgecolors='black', linewidth=1, marker='o')
plt.title('Input data')
# Розбиття даних на навчальний та тестовий набори
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=5)
# Класифікатор на основі гранично випадкових лісів
params = {'n_estimators': 100, 'max_depth': 4, 'random_state': 0}
if len(sys.argv) > 1:
    if sys.argv[1] == 'balance':
        params = {'n_estimators': 100, 'max_depth': 4, 'random_state': 0, 'class_weight': 'ba
    else:
        raise TypeError("Invalid input argument; should be 'balance'")
```

```
        raise TypeError("Invalid input argument; should be 'balance'")
classifier = ExtraTreesClassifier(**params)
classifier.fit(X_train, y_train)
visualize_classifier(classifier, X_train, y_train, 'Training dataset')
y_test_pred = classifier.predict(X_test)
visualize_classifier(classifier, X_test, y_test, 'Test dataset')
# Обчислення показників ефективності класифікатора
class_names = ['Class-0', 'Class-1']
print("\n" + "#" * 40)
print("\nClassifier performance on training dataset\n")
print(classification_report(y_train, classifier.predict(X_train), target_names=c
print("#" * 40 + "\n")
print("#" * 40)
print("\nClassifier performance on test dataset\n")
print(classification_report(y_test, y_test_pred, target_names=class_names))
print("#" * 40 + "\n")
plt.show()

```

Рис 3. Код програми файлу LR_4_task_2







Classifier performance on training dataset

	precision	recall	f1-score	support
Class-0	0.44	0.93	0.60	181
Class-1	0.98	0.77	0.86	944
accuracy			0.78	375
macro avg	0.72	0.84	0.73	375
weighted avg	0.88	0.78	0.80	375

Рис 4. Результат виконання коду файлу LR_4_task_2

Завдання 2.3. Знаходження оптимальних навчальних параметрів за допомогою сіткового пошуку.

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
import pandas as pd
from utilities import visualize_classifier
input_file = 'data_random_forests.txt'
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1]
# Розбиття даних на три класи на підставі міток
class_0 = np.array(X[y == 0])
class_1 = np.array(X[y == 1])
class_2 = np.array(X[y == 2])
# Розбиття даних на навчальний та тестовий набори
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=5)
# Визначення сітки значень параметрів
parameter_grid = [{'n_estimators': [100], 'max_depth': [2, 4, 7, 12, 16]},
                  {'max_depth': [4], 'n_estimators': [25, 50, 100, 250]}
                  ]
metrics = ['precision_weighted', 'recall_weighted']
for metric in metrics:
    print("\n##### Searching optimal parameters for", metric)
    classifier = GridSearchCV(
        ExtraTreesClassifier(random_state=0),
        parameter_grid, cv=5, scoring=metric)
    classifier.fit(X_train, y_train)
```

```python
classifier.fit(X_train, y_train)
df = pd.DataFrame(classifier.cv_results_)
df_columns_to_print = [column for column in df.columns if 'param' in column or 'score' in column]
print(df[df_columns_to_print])
print("\nBest parameters:", classifier.best_params_)
y_pred = classifier.predict(X_test)
print("\nPerformance report:\n")
print(classification_report(y_test, y_pred))
```

Рис 5. Код програми файлу LR_4_task_3

```
##### Searching optimal parameters for precision_weighted
   mean_score_time  std_score_time  ... std_test_score rank_test_score
0         0.011244    3.890568e-04  ...       0.025132              1
1         0.011416    5.022505e-04  ...       0.023032              5
2         0.011253    4.321109e-04  ...       0.026560              4
3         0.011960    1.387637e-05  ...       0.028334              8
4         0.012765    7.462774e-04  ...       0.033429              9
5         0.003592    4.895782e-04  ...       0.029698              2
6         0.005984    9.536743e-08  ...       0.020401              7
7         0.010372    4.885192e-04  ...       0.023032              5
8         0.035550    1.915573e-02  ...       0.026867              3

[9 rows x 13 columns]

Best parameters: {'max_depth': 2, 'n_estimators': 100}

Performance report:

              precision    recall  f1-score   support

         0.0       0.94      0.81      0.87        79
         1.0       0.81      0.86      0.83        70
         2.0       0.83      0.91      0.87        76

    accuracy                           0.86       225
   macro avg       0.86      0.86      0.86       225
weighted avg       0.86      0.86      0.86       225
```

```
##### Searching optimal parameters for recall_weighted
  mean_score_time  std_score_time  ... std_test_score rank_test_score
0        0.012699        0.001604  ...       0.027075               1
1        0.014285        0.004264  ...       0.022468               5
2        0.014895        0.001292  ...       0.026749               3
3        0.013389        0.000439  ...       0.028497               8
4        0.017204        0.004969  ...       0.034744               9
5        0.004139        0.000478  ...       0.029407               1
6        0.007354        0.001175  ...       0.020096               7
7        0.012357        0.001032  ...       0.022468               5
8        0.034883        0.009096  ...       0.026749               3

[9 rows x 13 columns]

Best parameters: {'max_depth': 2, 'n_estimators': 100}

Performance report:

              precision    recall  f1-score   support

         0.0       0.94      0.81      0.87        79
         1.0       0.81      0.86      0.83        70
         2.0       0.83      0.91      0.87        76

    accuracy                           0.86       225
   macro avg       0.86      0.86      0.86       225
weighted avg       0.86      0.86      0.86       225
```

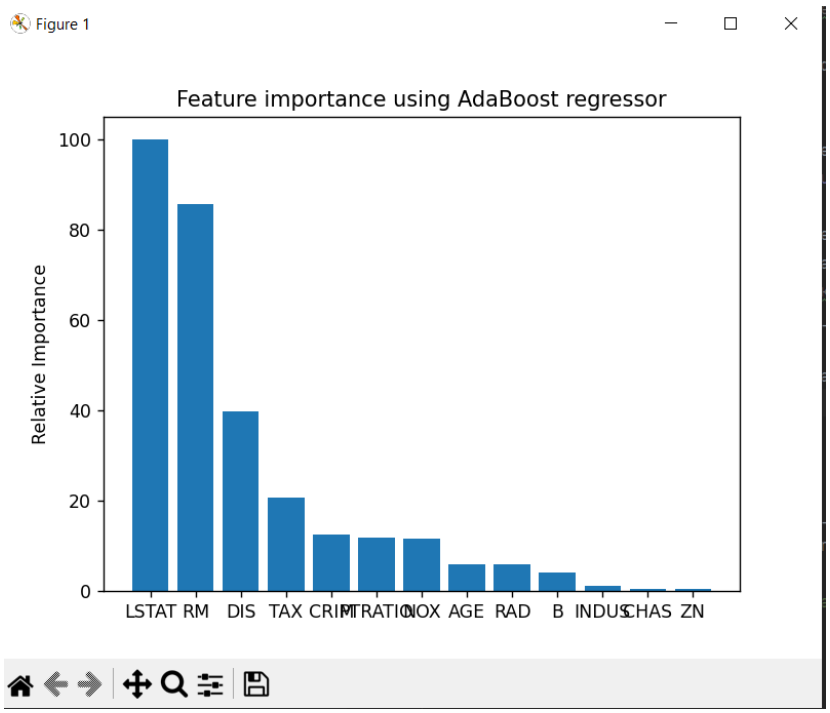
Рис 6. Результат виконання коду файлу LR_4_task_3

Завдання 2.4. Обчислення відносної важливості ознак.

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn import datasets
from sklearn.metrics import mean_squared_error, explained_variance_score
from sklearn.model_selection import train_test_split
from sklearn.utils import shuffle
# Завантаження даних із цінами на нерухомість
housing_data = datasets.load_boston()
# Перемішування даних
X, y = shuffle(housing_data.data, housing_data.target, random_state=7)
# Розбиття даних на навчальний та тестовий набори
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=7)
# Модель на основі регресора AdaBoost
regressor = AdaBoostRegressor(DecisionTreeRegressor(max_depth=4),
                              n_estimators=400, random_state=7)
regressor.fit(X_train, y_train)
# Обчислення показників ефективності регресора AdaBoost
y_pred = regressor.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
evs = explained_variance_score(y_test, y_pred)
print("\nADABOOST REGRESSOR")
print("Mean squared error =", round(mse, 2))
print("Explained variance score =", round(evs, 2))
# Вилучення важливості ознак
feature_importances = regressor.feature_importances_
feature_names = housing_data.feature_names
# Нормалізація значень важливості ознак
feature_importances = 100.0 * (feature_importances / max(feature_importances))
```

```
feature_importances = 100.0 * (feature_importances / max(feature_importances))
# Сортування та перестановка значень
index_sorted = np.flipud(np.argsort(feature_importances))
# Розміщення міток уздовж осі X
pos = np.arange(index_sorted.shape[0]) + 0.5
# Побудова стовпчастої діаграми
plt.figure()
plt.bar(pos, feature_importances[index_sorted], align='center')
plt.xticks(pos, feature_names[index_sorted])
plt.ylabel('Relative Importance')
plt.title('Feature importance using AdaBoost regressor')
plt.show()
```

Рис 7. Код програми файлу LR_4_task_4




```
ADABOOST REGRESSOR
Mean squared error = 22.7
Explained variance score = 0.79

Process finished with exit code 0
```

Рис 8. Результат виконання коду файлу LR_4_task_5

Завдання 2.5. Прогнозування інтенсивності дорожнього руху за допомогою класифікатора на основі гранично випадкових лісів.

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report, mean_absolute_error
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.metrics import classification_report
# Завантаження вхідних даних
input_file = 'traffic_data.txt'
data = []
with open(input_file, 'r') as f:
    for line in f.readlines():
        items = line[:-1].split(',')
        data.append(items)
data = np.array(data)
# Перетворення рядкових даних на числові
label_encoder = []
X_encoded = np.empty(data.shape)
for i, item in enumerate(data[0]):
    if item.isdigit():
        X_encoded[:, i] = data[:, i]
    else:
        label_encoder.append(preprocessing.LabelEncoder())
        X_encoded[:, i] = label_encoder[-1].fit_transform(data[:, i])
X = X_encoded[:, :-1].astype(int)
y = X_encoded[:, -1].astype(int)
# Розбиття даних на навчальний та тестовий набори
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=5)
# Регресор на основі гранично випадкових лісів
params = {'n_estimators': 100, 'max_depth': 4, 'random_state': 0}
regressor = ExtraTreesRegressor(**params)
regressor.fit(X_train, y_train)
# Обчислення характеристик ефективності регресора на тестових даних
y_pred = regressor.predict(X_test)
print("Mean absolute error:", round(mean_absolute_error(y_test, y_pred), 2))
# Тестування кодування на одиночному прикладі
test_datapoint = ['Saturday', '10:20', 'Atlanta', 'no']
test_datapoint_encoded = [-1] * len(test_datapoint)
count = 0
for i, item in enumerate(test_datapoint):
    if item.isdigit():
        test_datapoint_encoded[i] = int(test_datapoint[i])
    else:
        test_datapoint_encoded[i] = int(label_encoder[count].transform([test_datapoint[i]]))
        count = count + 1
test_datapoint_encoded = np.array(test_datapoint_encoded)
# Прогнозування результату для тестової точки даних
print("Predicted traffic:", int(regressor.predict([test_datapoint_encoded])[0]))
```

Рис 9. Код програми файлу LR_4_task_5

```
Mean absolute error: 7.42
Predicted traffic: 26

Process finished with exit code 0
```

Рис 10. Результат виконання коду файлу LR_4_task_5

Завдання 2.6. Створення навчального конвеєра (конвеєра машинного навчання).

```python
from sklearn.datasets import _samples_generator
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn.pipeline import Pipeline
from sklearn.ensemble import ExtraTreesClassifier
# Генерація даних
X, y = _samples_generator.make_classification(n_samples=150,
                                              n_features=25, n_classes=3, n_informative=6,
                                              n_redundant=0, random_state=7)
# Вибір k найважливіших ознак
k_best_selector = SelectKBest(f_regression, k=9)
# Ініціалізація класифікатора на основі гранично випадкового лісу
classifier = ExtraTreesClassifier(n_estimators=60, max_depth=4)
# Створення конвеєра
processor_pipeline = Pipeline([('selector', k_best_selector), ('erf', classifier)])
# Встановлення параметрів
processor_pipeline.set_params(selector__k=7, erf__n_estimators=30)
# Навчання конвеєра
processor_pipeline.fit(X, y)
# Прогнозування результатів для вхідних даних
output = processor_pipeline.predict(X)
print("\nPredicted output:\n", output)
# Виведення оцінки
print("\nScore:", processor_pipeline.score(X, y))
# Виведення ознак, відібраних селектором конвеєра
status = processor_pipeline.named_steps['selector'].get_support()
# Вилучення та виведення індексів обраних ознак
selected = [i for i, x in enumerate(status) if x]
print("\nIndices of selected features:", ', '.join([str(x) for x in selected]))
```

Рис 11. Код програми файлу LR_4_task_6

```
Predicted output:
 [1 2 2 0 2 0 2 1 0 1 1 2 1 0 2 2 1 0 0 1 0 2 1 1 2 2 0 0 1 0 1 2 1 0 2 2 1
 1 2 2 2 0 1 2 2 1 2 2 1 0 1 2 2 2 2 0 2 2 0 2 2 0 2 0 2 2 1 1 1 2 0 1 0 2
 0 0 1 0 2 0 0 1 2 2 0 0 0 0 2 2 2 1 2 0 2 1 2 2 0 0 1 1 1 1 2 2 0 2 0 1 1
 0 2 1 1 0 1 1 1 1 0 0 0 1 2 0 0 0 2 1 2 0 0 1 0 1 1 0 1 1 1 1 2 0 0 1 2 0
 2 2]

Score: 0.8866666666666667

Indices of selected features: 4, 7, 8, 12, 14, 17, 22

Process finished with exit code 0
```

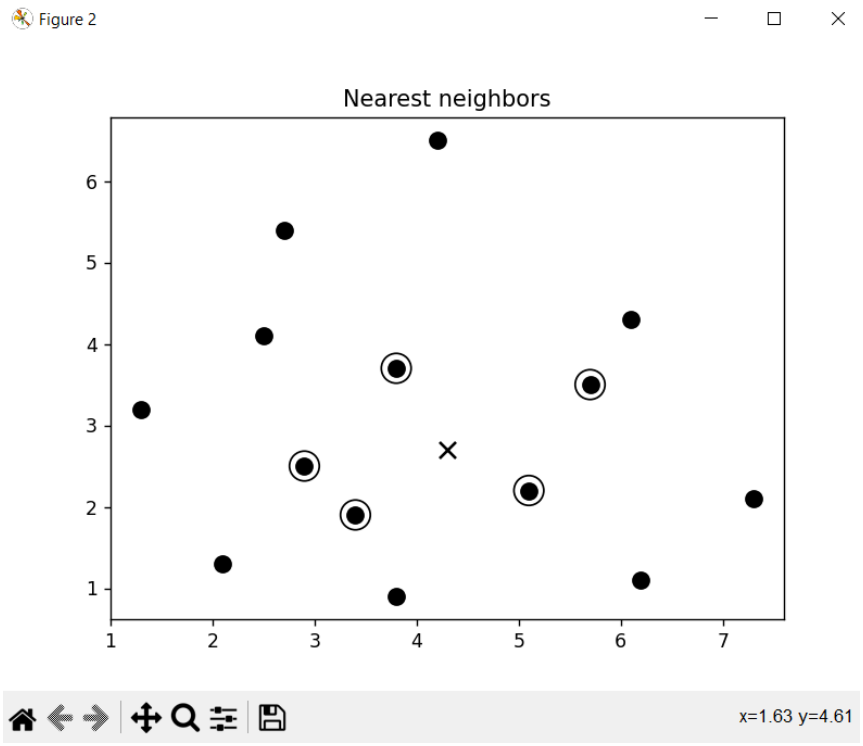
Рис 12. Результат виконання коду файлу LR_4_task_6

Висновок: Перший абзац містить прогнозовані вихідні мітки за допомогою конвеєра. Значення Score відображає ефективність конвеєра. Останній абзац містить індекси вибраних ознак.

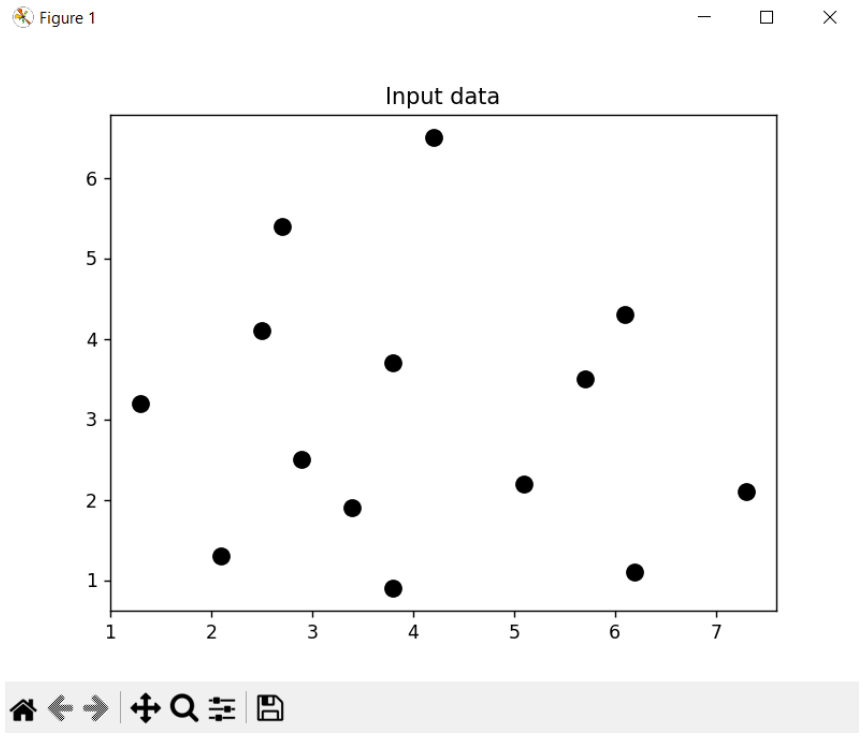
Завдання 2.7. Пошук найближчих сусідів.

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import NearestNeighbors
# Вхідні дані
X = np.array([[2.1, 1.3], [1.3, 3.2], [2.9, 2.5], [2.7, 5.4], [3.8, 0.9],
              [7.3, 2.1], [4.2, 6.5], [3.8, 3.7], [2.5, 4.1], [3.4, 1.9],
              [5.7, 3.5], [6.1, 4.3], [5.1, 2.2], [6.2, 1.1]])
# Кількість найближчих сусідів
k = 5
# Тестова точка даних
test_datapoint = [4.3, 2.7]
# Відображення вхідних даних на графіку
plt.figure()
plt.title('Input data')
plt.scatter(X[:, 0], X[:, 1], marker='o', s=75, color='black')
# Побудова моделі на основі методу k найближчих сусідів
knn_model = NearestNeighbors(n_neighbors=k, algorithm='ball_tree').fit(X)
distances, indices = knn_model.kneighbors([test_datapoint])
# Виведемо 'k' найближчих сусідів
print("\nK Nearest Neighbors:")
for rank, index in enumerate(indices[0][:k], start=1):
    print(str(rank) + " ==>", X[index])
# Візуалізація найближчих сусідів разом із тестовою точкою даних
plt.figure()
plt.title('Nearest neighbors')
plt.scatter(X[:, 0], X[:, 1], marker='o', s=75, color='k')
plt.scatter(X[indices][0][:][:, 0], X[indices][0][:][:, 1],
            marker='o', s=250, color='k', facecolors='none')
plt.scatter(test_datapoint[0], test_datapoint[1],
            marker='x', s=75, color='k')
plt.show()
```

Рис 13. Код програми файлу LR_4_task_7



вхідні дані, тестову точку і її 5 найближчих сусідів



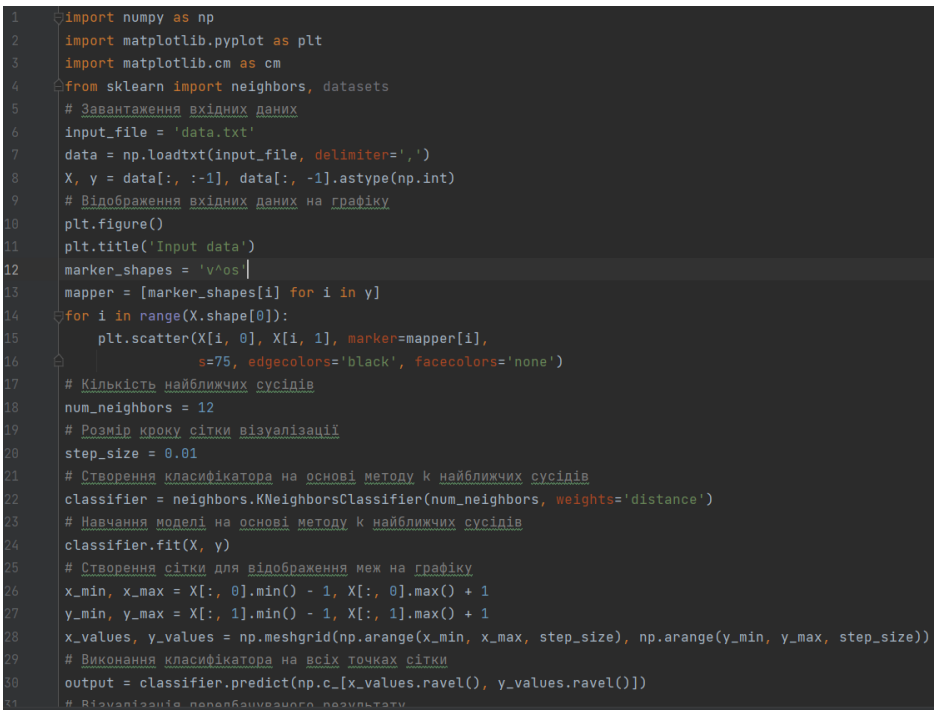
вхідні дані

```
K Nearest Neighbors:
1 ==> [5.1 2.2]
2 ==> [3.8 3.7]
3 ==> [3.4 1.9]
4 ==> [2.9 2.5]
5 ==> [5.7 3.5]

Process finished with exit code 0
```

5 найближчих сусідів

Рис 14. Результат виконання коду файлу LR_4_task_7

Завдання 2.8. Створити класифікатор методом k найближчих сусідів.

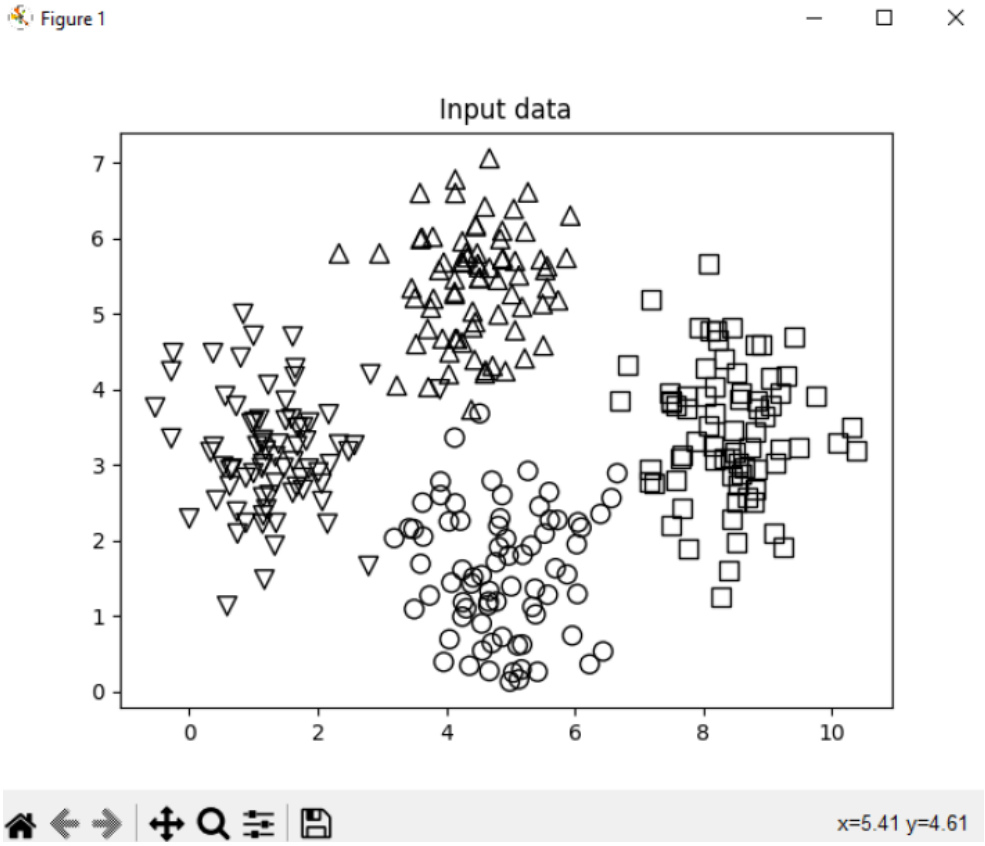


```python
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from sklearn import neighbors, datasets
# Завантаження вхідних даних
input_file = 'data.txt'
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1].astype(np.int)
# Відображення вхідних даних на графіку
plt.figure()
plt.title('Input data')
marker_shapes = 'v^os'
mapper = [marker_shapes[i] for i in y]
for i in range(X.shape[0]):
    plt.scatter(X[i, 0], X[i, 1], marker=mapper[i],
                s=75, edgecolors='black', facecolors='none')
# Кількість найближчих сусідів
num_neighbors = 12
# Розмір кроку сітки візуалізації
step_size = 0.01
# Створення класифікатора на основі методу k найближчих сусідів
classifier = neighbors.KNeighborsClassifier(num_neighbors, weights='distance')
# Навчання моделі на основі методу k найближчих сусідів
classifier.fit(X, y)
# Створення сітки для відображення меж на графіку
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
x_values, y_values = np.meshgrid(np.arange(x_min, x_max, step_size), np.arange(y_min, y_max, step_size))
# Виконання класифікатора на всіх точках сітки
output = classifier.predict(np.c_[x_values.ravel(), y_values.ravel()])
# Візуалізація перерахованого результату
```

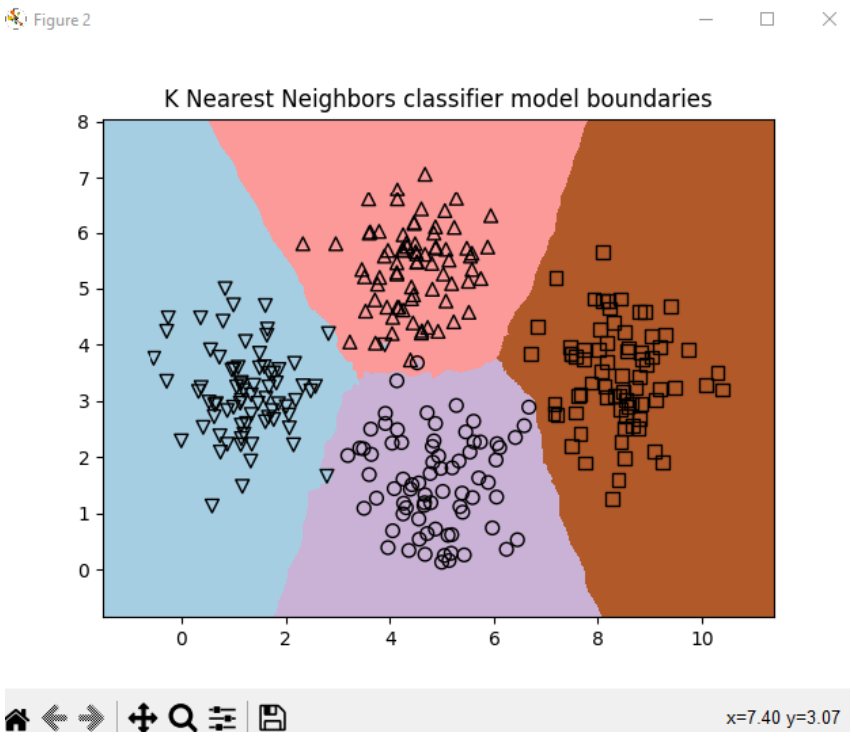
```
    # Візуалізація передбачуваного результату
    output = output.reshape(x_values.shape)
    plt.figure()
    plt.pcolormesh(x_values, y_values, output, cmap=cm.Paired)
    # Накладання навчальних точок на карту
    for i in range(X.shape[0]):
        plt.scatter(X[i, 0], X[i, 1], marker=mapper[i],
                    s=50, edgecolors='black', facecolors='none')
    plt.xlim(x_values.min(), x_values.max())
    plt.ylim(y_values.min(), y_values.max())
    plt.title('K Nearest Neighbors classifier model boundaries')
    # Тестування вхідної точки даних
    test_datapoint = [5.1, 3.6]
    plt.figure()
    plt.title('Test datapoint')
    for i in range(X.shape[0]):
        plt.scatter(X[i, 0], X[i, 1], marker=mapper[i],
                    s=75, edgecolors='black', facecolors='none')
    plt.scatter(test_datapoint[0], test_datapoint[1], marker='x',
                linewidth=6, s=200, facecolors='black')
    # Вилучення К найближчих сусідів
    _, indices = classifier.kneighbors([test_datapoint])
    indices = indices.astype(np.int)[0]
    # Відображення К найближчих сусідів на графіку
    plt.figure()
    plt.title('K Nearest Neighbors')
    for i in indices:
        plt.scatter(X[i, 0], X[i, 1], marker=mapper[y[i]],
                    linewidth=3, s=100, facecolors='black')
    plt.scatter(test_datapoint[0], test_datapoint[1], marker='x',
```

```
    # Відображення К найближчих сусідів на графіку
    plt.figure()
    plt.title('K Nearest Neighbors')
    for i in indices:
        plt.scatter(X[i, 0], X[i, 1], marker=mapper[y[i]],
                    linewidth=3, s=100, facecolors='black')
    plt.scatter(test_datapoint[0], test_datapoint[1], marker='x',
                linewidth=6, s=200, facecolors='black')
    for i in range(X.shape[0]):
        plt.scatter(X[i, 0], X[i, 1], marker=mapper[i],
                    s=75, edgecolors='black', facecolors='none')
    print("Predicted output:", classifier.predict([test_datapoint])[0])
    plt.show()
```

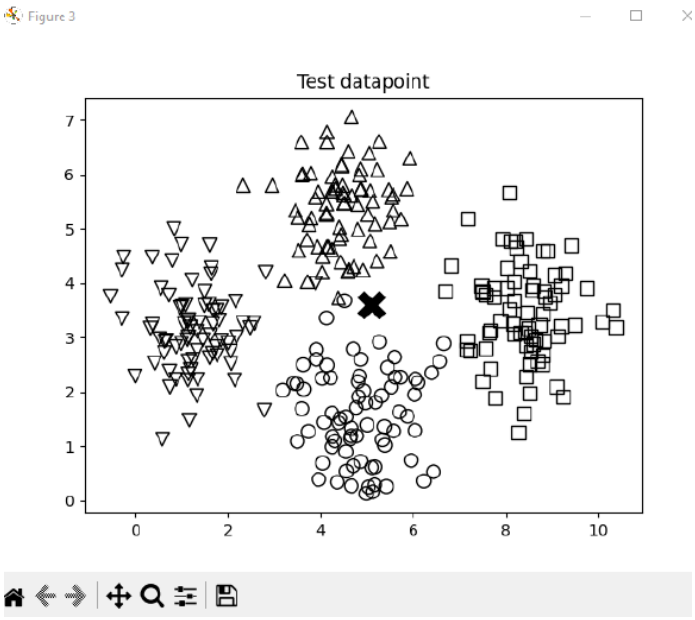
Рис 15. Код програми файлу LR_4_task_8



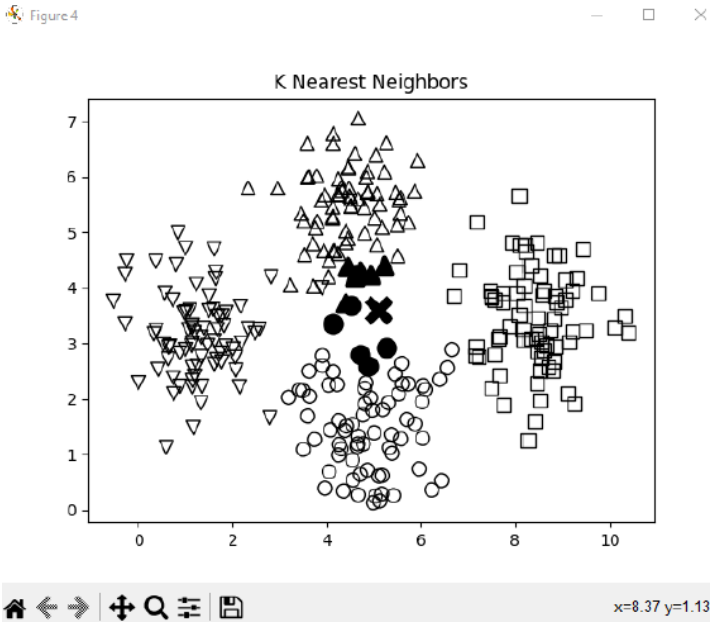
вхідні дані



межі класифікатора



тестова точка до вхідного набору даних



12 найближчих сусідів

Рис 16. Результат виконання коду файлу LR_4_task_8

Завдання 2.9. Обчислення оцінок подібності.

```python
import argparse
import json
import numpy as np
def build_arg_parser():
    parser = argparse.ArgumentParser(description='Compute similarity score')
    parser.add_argument('--user1', dest='user1', required=True, help='First user')
    parser.add_argument('--user2', dest='user2', required=True,
                        help='Second user')
    parser.add_argument("--score-type", dest="score_type", required=True,
                        choices=['Euclidean', 'Pearson'], help='Similarity metric to be used')
    return parser
# Обчислення оцінки евклідова відстані між користувачами user1 та user2
def euclidean_score(dataset,user1,user2):
    if user1 not in dataset:
        raise TypeError('Cannot find ' + user1 + ' in the dataset')

    if user2 not in dataset:
        raise TypeError('Cannot find ' + user2 + ' in the dataset')
    # Фільми, оцінені обома користувачами, user1 та user2
    common_movies = {}
    for item in dataset[user1]:
        if item in dataset[user2]:
            common_movies[item] = 1
    # За відсутності фільмів, оцінених обома користувачами, оцінка приймається рівною 0
    if len(common_movies) == 0:
        return 0
    squared_diff = []
    for item in dataset[user1]:
        if item in dataset[user2]:
            squared_diff.append(np.square(dataset[user1][item] - dataset[user2][item]))
    return 1 / (1 + np.sqrt(np.sum(squared_diff)))
```

```python
# Обчислення кореляційної оцінки Пірсона між користувачем1 і користувачем2
def pearson_score(dataset, user1, user2):
    if user1 not in dataset:
        raise TypeError('Cannot find ' + user1 + ' in the dataset')
    if user2 not in dataset:
        raise TypeError('Cannot find ' + user2 + ' in the dataset')
    # Фільми, оцінені обома користувачами, user1 та user2
    common_movies = {}
    for item in dataset[user1]:
        if item in dataset[user2]:
            common_movies[item] = 1
    num_ratings = len(common_movies)
    # За відсутності фільмів, оцінених обома користувачами, оцінка приймається рівною 0
    if num_ratings == 0:
        return 0
    # Обчислення суми рейтингових оцінок усіх фільмів, оцінених обома користувачами
    user1_sum = np.sum([dataset[user1][item] for item in common_movies])
    user2_sum = np.sum([dataset[user2][item] for item in common_movies])
    # Обчислення Суми квадратів рейтингових оцінок всіх фільмів, оцінених обома кори-стувачами
    user1_squared_sum = np.sum([np.square(dataset[user1][item]) for item in common_movies])
    user2_squared_sum = np.sum([np.square(dataset[user2][item]) for item in common_movies])
    # Обчислення суми творів рейтингових оцінок всіх фільмів, оцінених обома користува-чами
    sum_of_products = np.sum([dataset[user1][item] * dataset[user2][item] for item in common_movies])
    # Обчислення коефіцієнта кореляції Пірсона
    Sxy = sum_of_products - (user1_sum * user2_sum / num_ratings)
    Sxx = user1_squared_sum - np.square(user1_sum) / num_ratings
    Syy = user2_squared_sum - np.square(user2_sum) / num_ratings
    if Sxx * Syy == 0:
        return 0
    return Sxy / np.sqrt(Sxx * Syy)
```

```python
if __name__ == '__main__':
    args = build_arg_parser().parse_args()
    user1 = args.user1
    user2 = args.user2
    score_type = args.score_type
    ratings_file = 'ratings.json'
    with open(ratings_file, 'r') as f:
        data = json.loads(f.read())
    if score_type == 'Euclidean':
        print("\nEuclidean score:")
        print(euclidean_score(data, user1, user2))
    else:
        print("\nPearson score:")
        print(pearson_score(data, user1, user2))

```

Рис 17. Код програми файлу LR_4_task_9

```
earson score:
.9909924304103233
S C:\Users\Admin\PycharmProjects\labka4> python LR_4_task_9.py --user1 "David Smith" --user2 "Bill Duffy" --score-type Euclidean

uclidean score:
.585786437626905
S C:\Users\Admin\PycharmProjects\labka4> python LR_4_task_9.py --user1 "David Smith" --user2 "Bill Duffy" --score-type Pearson

earson score:
.9909924304103233
S C:\Users\Admin\PycharmProjects\labka4>
```

```
Pearson score:
-0.7236759610155113
PS C:\Users\Admin\PycharmProjects\labka4> python LR_4_task_9.py --user1 "David Smith" --user2 "Samuel Miller" --score-type Euclidean

Euclidean score:
0.30303243470068705
PS C:\Users\Admin\PycharmProjects\labka4> python LR_4_task_9.py --user1 "David Smith" --user2 "Samuel Miller" --score-type Pearson

Pearson score:
0.7587869106393281
PS C:\Users\Admin\PycharmProjects\labka4> python LR_4_task_9.py --user1 "David Smith" --user2 "Julie Hammel" --score-type Euclidean

Euclidean score:
0.2857142857142857
PS C:\Users\Admin\PycharmProjects\labka4> python LR_4_task_9.py --user1 "David Smith" --user2 "Julie Hammel" --score-type Pearson

Pearson score:
0
PS C:\Users\Admin\PycharmProjects\labka4> python LR_4_task_9.py --user1 "David Smith" --user2 "Clarrisa Jackson" --score-type Euclidean

Euclidean score:
Traceback (most recent call last):
  File "C:\Users\Admin\PycharmProjects\labka4\LR_4_task_9.py", line 100, in <module>
    print(euclidean_score(data, user1, user2))
  File "C:\Users\Admin\PycharmProjects\labka4\LR_4_task_9.py", line 22, in euclidean_score
    raise TypeError('Cannot find ' + user2 + ' in the dataset')
TypeError: Cannot find Clarrisa Jackson in the dataset
PS C:\Users\Admin\PycharmProjects\labka4> python LR_4_task_9.py --user1 "David Smith" --user2 "Clarissa Jackson" --score-type Euclidean

Euclidean score:
0.2098979685563564
PS C:\Users\Admin\PycharmProjects\labka4> python LR_4_task_9.py --user1 "David Smith" --user2 "Clarissa Jackson" --score-type Pearsonn
usage: LR_4_task_9.py [-h] --user1 USER1 --user2 USER2 --score-type {Euclidean,Pearson}
LR_4_task_9.py: error: argument --score-type: invalid choice: 'Pearsonn' (choose from 'Euclidean', 'Pearson')
PS C:\Users\Admin\PycharmProjects\labka4> python LR_4_task_9.py --user1 "David Smith" --user2 "Clarissa Jackson" --score-type Pearson

Pearson score:
0.6944217062199275
PS C:\Users\Admin\PycharmProjects\labka4> python LR_4_task_9.py --user1 "David Smith" --user2 "Adam Cohen" --score-type Euclidean

Euclidean score:
0.3874258867227930
PS C:\Users\Admin\PycharmProjects\labka4> python LR_4_task_9.py --user1 "David Smith" --user2 "Adam Cohen" --score-type Pearson

Pearson score:
0.9081082738950217
PS C:\Users\Admin\PycharmProjects\labka4> python LR_4_task_9.py --user1 "David Smith" --user2 "Chris Duncan" --score-type Euclidean

Euclidean score:
0.3874258867227930
PS C:\Users\Admin\PycharmProjects\labka4> python LR_4_task_9.py --user1 "David Smith" --user2 "Chris Duncan" --score-type Pearson

Pearson score:
1.0
PS C:\Users\Admin\PycharmProjects\labka4>
```

Рис 18. Результат виконання коду файлу LR_4_task_9

Оцінка подібності за Пірсоном демонструє кращі результати в порівнянні з евклідовою оцінкою подібності.

Завдання 2.10. Пошук користувачів зі схожими уподобаннями методом колаборативної фільтрації.

```python
import argparse
import json
import numpy as np
from LR_4_task_9m import pearson_score
def build_arg_parser():
    parser = argparse.ArgumentParser(description='Find users who are similar to the in-put user')
    parser.add_argument('--user', dest='user', required=True,
                        help='Input user')
    return parser
# Знаходження користувачів у наборі даних, схожих на введеного користувача
def find_similar_users(dataset, user, num_users):
    if user not in dataset:
        raise TypeError('Cannot find ' + user + ' in the dataset')
    # Обчислення оцінки подібності за Пірсоном між
    # вказаним користувачем та всіма іншими
    # користувачами в наборі даних
    scores = np.array([[x, pearson_score(dataset, user,
                                         x)] for x in dataset if x != user])
    # Сортування оцінок за спаданням
    scores_sorted = np.argsort(scores[:, 1])[::-1]
    # Вилучення оцінок перших 'num_users' користувачів
    top_users = scores_sorted[:num_users]
    return scores[top_users]
if __name__ == '__main__':
    args = build_arg_parser().parse_args()
    user = args.user
    ratings_file = 'ratings.json'
    with open(ratings_file, 'r') as f:
        data = json.loads(f.read())
    print('\nUsers similar to ' + user + ':\n')
    similar_users = find_similar_users(data, user, 3)
```

```
    print('User\t\t\tSimilarity score')
    print('-' * 41)
    for item in similar_users:
        print(item[0], '\t\t', round(float(item[1]), 2))
```

Рис 19. Код програми файлу LR_4_task_10

```
LR_4_task_10.py: error: the following arguments are required: --user
PS C:\Users\Admin\PycharmProjects\labka4> python LR_4_task_10.py --user "Clarissa Jackson"

Users similar to Clarissa Jackson:

User                     Similarity score
-----------------------------------------
Chris Duncan              1.0
Bill Duffy                0.83
Samuel Miller             0.73
PS C:\Users\Admin\PycharmProjects\labka4> python LR_4_task_10.py --user "Bill Duffy"

Users similar to Bill Duffy:

User                     Similarity score
-----------------------------------------
David Smith               0.99
Samuel Miller             0.88
Adam Cohen                0.86
PS C:\Users\Admin\PycharmProjects\labka4>
```

Рис 20. Результат виконання коду файлу LR_4_task_10

Юзер “Clarissa Jackson” має одинакові вподобання з користувачем “Chris Duncan”, а користувач “Bill Duffy” – майже однакові з “David Smith”.

Завдання 2.11. Створення рекомендаційної системи фільмів.

```python
import argparse
import json
import numpy as np
from LR_4_task_9m import pearson_score
from LR_4_task_10m import find_similar_users
def build_arg_parser():
    parser = argparse.ArgumentParser(description='Find the movie recommendations for the given user')
    parser.add_argument('--user', dest='user', required=True,
                        help='Input user')
    return parser
# Отримання рекомендації щодо фільмів для вказаного користувача
def get_recommendations(dataset, input_user):
    if input_user not in dataset:
        raise TypeError('Cannot find ' + input_user + ' in the dataset')
    overall_scores = {}
    similarity_scores = {}
    for user in [x for x in dataset if x != input_user]:
        similarity_score = pearson_score(dataset, input_user, user)
        if similarity_score <= 0:
            continue
        filtered_list = [x for x in dataset[user] if x not in \
                         dataset[input_user] or dataset[input_user][x] == 0]
        for item in filtered_list:
            overall_scores.update({item: dataset[user][item] * similarity_score})
            similarity_scores.update({item: similarity_score})
    if len(overall_scores) == 0:
        return ['No recommendations possible']
    # Генерація рейтингів фільмів за допомогою їх нормалізації
    movie_scores = np.array([[score / similarity_scores[item], item]
                             for item, score in overall_scores.items()])
```

```
    movie_recommendations = [movie for _, movie in movie_scores]
    return movie_recommendations
if __name__ == '__main__':
    args = build_arg_parser().parse_args()
    user = args.user
    ratings_file = 'ratings.json'
    with open(ratings_file, 'r') as f:
        data = json.loads(f.read())
    print("\nMovie recommendations for " + user + ":")
    movies = get_recommendations(data, user)
    for i, movie in enumerate(movies):
        print(str(i + 1) + '. ' + movie)
```

Рис 21. Код програми файлу LR_4_task_11

```
PS C:\Users\Admin\PycharmProjects\labka4> python LR_4_task_11.py --user "Julie Hammel"

Movie recommendations for Julie Hammel:
1. The Apartment
2. Vertigo
3. Raging Bull
PS C:\Users\Admin\PycharmProjects\labka4> python LR_4_task_11.py --user "Clarissa Jackson"

Movie recommendations for Clarissa Jackson:
1. No recommendations possible
PS C:\Users\Admin\PycharmProjects\labka4>
```

Рис 22. Результат виконання коду файлу LR_4_task_11

Julie Hammel = 3 реки

Кларіси = 0

Висновок: в даній лабораторній роботі навчився використовувати спеціалізовані бібліотеки та мову програмування Python, досліджувати методи ансамблів у машинному навчанні та створювати рекомендаційні системи.

Посилання на гіт: https://github.com/Max1648/Artificial-Intelligence2.